REMARKS

Claims 22 to 48 were pending when last examined; claims 45 to 48 were withdrawn from consideration. Applicant has amended claims 22, 23, 31, 33 to 35, 41, 43, and 44, and canceled claims 27 to 30 and 37 to 40.

Interview Summary

Applicant thanks the Examiner and his supervisory for the in-person interview that occurred on January 17, 2006. Applicant notes one discrepancy in the Interview Summary. The Examiner wrote, "The issue appears to lie in the level of dependence between the GUI Controller (which sends raw, non-graphical data) and the Embedded Controller (which uses local graphics data to provide a graphic depiction of the raw data on the display)." January 17, 2006 Interview Summary, p. 1.

In the above quoted sentence, the Examiner inverted the roles of the GUI controller and the embedded controller. The GUI controller and the embedded controller communicate raw, non-graphical data with each other. However, it is the GUI controller that uses local resources (e.g., executable codes) to provide a graphical depiction of the raw, non-graphical data on a display.

Claim Rejections

The Examiner rejected claims 22 to 44 under 35 U.S.C. § 103(a) as being unpatentable over U.S. Patent Nos. 5,995,120 ("Dye") and 6,118,462 ("Margulis").

Claim 22

Dye teaches away from the proposed modification and the proposed modification changes Dye's principle of operation

The Examiner found that Dye does not disclose separate memories for the CPU and the integrated memory controller (IMC) but it would be obvious to modify Dye in view of Margulis to provide separate memories for the CPU and the IMC.

> Dye teaches a graphics controller that is embedded in a memory controller, but doesn't specifically teach distinct system and graphics controller each having their own controller specific memory. Margulis teaches a system using a graphics controller, similar to that of Dye, but further teaches, in column 2, line

> 62 through column 3, line 30 and in figure 2, a system with distinct system and graphics controllers each having their own controller specific memory. It would have been obvious to one of ordinary skill in the art, having the teachings of Dye and Margulis before him at the time the invention was made to modify the controller system of Dye, to separate the two controllers. One would have been motivated to make such a combination because Dye teaches two similar controllers, but combines them to minimize memory transfers, similar to that of Margulis.

September 30, 2005 Office Action, pp. 3 and 4 (emphasis added). Applicant respectfully traverses.

Dye specifically discloses providing a single system memory for the CPU and the IMC in order to save data transfers between otherwise separate memories for the CPU and the graphical controller in the prior art. "The IMC also improves overall system performance and response using main system memory for graphical information and storage." Dye, col. 3, lines 10 to 12. Furthermore,

> In many cases, the graphical data is not required to leave the system memory110 and is not required to move to another location in system memory 110, but rather the display list-based operation and high level graphical protocol of the IMC 140 of the present invention enables the CPU 102 to instruct the IMC 104 how window and other graphical data is presented on the screen.

Dye, col. 11, lines 28 to 36. Thus, Dye teaches against providing separate memories for the CPU and the IMC. Modifying Dye to have separate memories for the CPU and the IMC would also change the operating principles of Dye since additional data transfers would be necessary between the CPU, the IMC, and their memories for graphical operations. Accordingly, amended claim 22 is patentable over the combination of Dye and Margulis because Dye teaches away from the modification suggested by the Examiner and such modification would change Dye's principle of operation.

Dye and Margulis do not disclose all the elements of amended claim 22

Addressing Applicant's arguments regarding claim 22 from the July 21, 2005 Response to Final Office Action, the Examiner stated:

> In response, the examiner respectfully submits that Dye teaches, in column 3, lines 5-20, in column 2, lines 50-67 and column 1, lines 46-61, a graphics controller comprised in an integrated memory controller (IMC) which includes graphic processing capabilities to relieve workload from the main CPU. This shows the same level of independence as is presented in the claimed invention. This can be shown by the claimed invention teaching a reliance on the

> embedded processor for supplying the status and the use of this status information for rendering, and further provides video outputs.

September 30, 2005 Office Action, p. 10, ¶ 24 (emphasis added). Furthermore, the Examiner states:

> In response, the examiner respectfully submits that Dye teaches, in column 21, lines 27-36, the system accessing supplemental libraries that contain application data such as 2-D and 3-D constraints, number of bits per pixel, and area, which a window works. The libraries are still believed to provide code that helps in the rendering of the objects appearances and functions by defining attributes, whether this is to further define or to do the initial definition.

September 30, 2005 Office Action, p. 10, ¶ 28 (emphasis added). Applicant respectfully traverses.

Applicant has amended independent claims 22, which now recites:

> 22. A control system for a device, wherein a graphic user interface ("GUI") controller operates a GUI of the device independently from an embedded controller of the device, the control system comprising:
>
> the embedded controller for controlling and monitoring the device;
>
> a liquid crystal display ("LCD") for displaying the GUI to a user, the GUI including:
>
> a first GUI object comprising a graphical presentation of a status of the device; and
>
> a second GUI object comprising a graphical presentation of a command to the device;
>
> a touch screen for detecting the command from the user;
>
> the GUI controller, comprising:
>
> at least one memory, comprising:
>
> a document buffer storing a document defining an appearance of the GUI, the document comprising:
>
> a first operation code ("opcode") identifying the first GUI object;
>
> a second opcode identifying a source of the status;
>
> a third opcode identifying the second GUI object; and

a fourth opcode identifying a destination of the command;

a data buffer storing the status and the command;

a GUI object library storing:

a first set of executable codes defining an appearance and a functionality of the first GUI object, the first set of executable codes comprising instructions for rendering the first GUI object, receiving non-graphical data of the status from the embedded controller, and further rendering the first GUI object to show a visual response to a change to the status;

a second set of executable codes defining an appearance and a functionality of the second GUI object, the second set of executable codes comprising instructions for rendering the second GUI object, receiving the command from the touch screen, further rendering the second GUI object to show a visual response to a change to the command, and sending non-graphical data of the command to the embedded controller;

a frame buffer storing at least one complete display frame image of the GUI;

a GUI processor for rendering the GUI and handling user inputs independently from the embedded controller, wherein:

the GUI processor is coupled to the touch screen via a touch screen analog to digital converter to receive the command from the user, wherein the touch screen is not directly connected to the embedded controller;

the GUI processor is coupled to the embedded controller via a serial UART interface to send the non-graphical data of the command to the embedded controller and to receive the non-graphical data of the status from the embedded controller;

the GUI processor is coupled to the at least one memory via a memory bus interface, wherein:

> in response to the first and the second opcodes, the GUI processor executes the first set of executable codes to render the first GUI object to the frame buffer independently from the embedded controller, to receive the non-graphical data of the status from the embedded controller, and to further render the first GUI object to the frame buffer to show a visual response to a change to the status independently from the embedded controller;
>
> in response to the third and the fourth opcodes, the GUI processor executes the second set of executable codes to render the second GUI object to the frame buffer independently from the embedded controller, to receive the command from the touch screen independently from the embedded controller, to further render the second GUI object to the frame buffer to show a visual response to a change to the command independently from the embedded controller, and to send the non-graphical data of the command to the embedded controller;

a pixel serializer coupled to the LCD to continuously refresh the LCD with the complete display frame image in the frame buffer that contains both the rendered first GUI object and the rendered second GUI object.

Amended claim 22 (emphasis added).

GUI controller operates independently from the embedded controller

Dye does not disclose a GUI controller with a GUI processor that executes codes (e.g., first and second sets of executable codes) to determine on its own how to graphically present status and command of a device independently from an embedded controller that controls the device as recited in amended claim 22. As recited in amended claim 22, the GUI controller executes the first and the second sets of executable codes to determine on its own how to graphically present the status and

the command independently from the embedded controller. Furthermore, amended claim 22 recites that only non-graphical data of the status and the command are communicated between the GUI controller and the embedded controller.

On the other hand, Dye discloses that the CPU executes codes and then sends graphical commands to instruct the IMC how to graphically present the windows/objects.

> Video screen changes or screen updates are preferably performed using the following operations. First, in response to software executing on the host CPU, such as applications software, the video driver executing on the CPU generates a video driver instruction list which includes screen update and/or graphics information for displaying video data on the screen. The video driver instruction list is provided to the Execution Engine in the graphics controller or IMC. The Execution Engine examines the video driver instruction list and generates a list of graphics and/or memory commands to the Graphics Engine. Thus the Execution Engine constructs a complete list of graphics or memory operations to be performed in response to desired screen change information.

Dye, col. 3, line 62 to col. 4, line 7 (emphasis added).

> The CPU 102 begins program execution by reading the recently decompressed program code from the system memory 110. Portions of the program code contain information necessary to write data and/or instructions back to the IMC 140 using a special graphical protocol according to the present invention to direct the IMC 140 to control the display output on the video display 142. In many cases, the graphical data is not required to leave the system memory 110 and is not required to move to another location in system memory 110, but rather the display list-based operation and high level graphical protocol of the IMC 140 of the present invention enables the CPU 102 to instruct the IMC 104 how window and other graphical data is presented on the screen. This provides a tremendous improvement over prior art systems.
>
> .... Instead, a computer system incorporating an IMC 140 according to the present invention includes a high level graphic protocol whereby the CPU 102 instructs the IMC 140 to manipulate the data stored in the system memory 110. For example, when text which appears in a window on the display screen is manipulated, the text is not required to leave the system memory 110 for processing by the CPU 102. Rather, the IMC 140 reads the text data into the system memory 110, preferably in ASCII format, and the IMC 140 processes the text data for display output. This operation is performed under the direction of the CPU 102 through the high level graphic protocol used by the IMC 140, as described further below. ....

Dye, col. 11, lines 22 to 53 (emphasis added).

> A video driver executes on the CPU 102 and generates a video driver instruction list which includes video display information regarding desired changes to the video output of the video monitor 142. The video display information includes screen update and/or graphics information for displaying video data on the display screen of the video monitor 142. For example, the video display information may include commands to draw a 3D texture map, or to bit blit pixel data from a first xy memory location to a second xy memory location, to render a polygon, or to assemble a display refresh list.
>
> The video driver instruction list is provided to the Execution Engine 210 in the graphics controller or IMC. The Execution Engine 210 examines the video driver instruction list and generates a list of graphics and/or memory commands to the Graphics Engine 212. Thus the Execution Engine 210 constructs a complete list of graphics or memory operations to be performed in response to desired screen change information. In response to an Assemble Display Refresh List command, the Execution Engine 210 assembles or constructs a display refresh list.

Dye, col. 21, lines 6 to 26 (emphasis added). As can be seen from above, it is the CPU that executes codes to determine how to graphically present windows/objects. The CPU then sends graphical commands to instruct the IMC how to graphically present the windows/objects. Thus, the IMC does not execute any codes to determine on its own how to graphically present the windows/object independently from the CPU while the recited GUI controller executes codes to determine on its own how to graphically present the status and the command independently from the recited embedded controller. Furthermore, the IMC communicates graphical commands with the CPU while the recited GUI controller only communicates non-graphical data of the status and the command with the recited embedded controller.

The Examiner cited col. 3, lines 5 to 20, col. 2, lines 50 to 67, and col. 1, lines 46 to 61 to show that Dye discloses the same level of independence between the IMC and the CPU as the recited GUI controller and the recited embedded controller. Col. 3, lines 5 to 20 of Dye describe the IMC architecture where the graphical data reside in the main memory and the IMC transfers data between the system bus and the system memory and also between the system memory and the video display output. Col. 2, lines 50 to 67 of Dye describe that the IMC performs pointer-based display list video operations. Col. 1, lines 46 to 61 of Dye describe a dedicated video processor that reduces the workload of the main CPU. While the above cited lines describe ways for a graphic controller to assist the CPU with graphics, they do not describe that the graphic controller decides on its own how to graphically present status and command independently from the CPU and that only non-

graphical data of the status and the command are communicated between the graphic controller and the CPU as recited in amended claim 22.

GUI object library that defines the graphical presentation of data

Amended claim 22 recites a GUI object library with GUI objects having executable codes that define the graphical presentations of status and command, and that the GUI processor executes those executable codes to render the graphical presentations of the status and the command. On the other hand, Dye discloses that the CPU defines the graphical presentations of the windows/objects and then uses supplemental libraries to translate the graphical presentations into graphical commands to the IMC.

> FIG. 7A illustrates operation of the software drivers which interface to the IMC 140. Essentially, each application requires a different set of constraints, such as whether the application is a 2-D or a 3-D application, the number of bits per pixel, the area in which the window works, and the capabilities of the subsystem. Based on the requirements of the application, the drivers make calls to supplemental libraries, such as 3-D protocols, compression and/or decompression libraries, and possibly a window assembly library, among others, to perform desired operations.

Dye, col. 21, lines 27 to 36. Fig. 7A shows that software drivers on the CPU takes what the application requires to be displayed and then uses the supplemental libraries to construct graphical commands in the proper protocol for the IMC. The IMC then follows these graphical commands to render the graphical presentations of the windows/objects. Thus, the IMC does not execute the supplemental libraries and the supplemental libraries do not define the graphical presentations of any status and command.

Touch screen coupled to the GUI controller but not the embedded controller

Dye does not disclose a touch screen that is coupled to a GUI controller but not the embedded controller as recited by amended claim 22. Dye simply does not disclose a touch screen. More importantly, Dye does not disclose any input device that is coupled to the IMC. Instead, the input devices are coupled by a system bus to communicate with the CPU. Although the IMC is also located on the same system bus, Dye does not disclose any input device that communicates with the IMC through the system bus. Amended claim 22 clearly recites that the touch screen communicates with the GUI controller through their coupling.

Margulis does not cure the deficiencies of Dye

Margulis does not cure the above deficiencies of Dye as recited against amended claim 22. Accordingly, amended claim 22 is patentable over the combination of Dye and Margulis because they do not disclose all the elements of amended claim 22.

Dye and Margulis are non-analogous art

Dye and Margulis are non-analogous art that are not in the field of Applicant's endeavor. Both Dye and Margulis relate to general purpose programmable computers with central processing units. See Dye, col. 2, line 56 to col. 3, line 20; Margulis, col. 3, lines 15 to 31. On the other hand, the claimed invention is related to "embedded systems, i.e. other than general purpose programmable computers." Specification, p. 1. This is reflected in amended claim 22, which recites an "embedded controller for controlling and monitoring the device." Amended claim 22.

Dye and Margulis are not reasonably pertinent to the particular problem with which the Applicant is concerned. Dye is concerned with how to increase the performance of computer systems by minimizing data movement and video data manipulation for video display updates. Dye, col. 2, lines 33 to 55. Margulis is concerned with how to provide memory architecture with a common memory that can be used for display memory and main memory without impairing the performance of the computer systems. Margulis, col. 3, lines 7 to 18. On the other hand, Applicant is concerned with how to reduce development time and engineering cost for customizing embedded processor firmware for different applications and I/O devices. See Specification, p. 2, line 24 to p. 3, line 16. One skilled in the art would not find Dye and Margulis reasonably pertinent to the particular problem with which the Applicant is concerned. Accordingly, Dye and Margulis cannot be cited against amended claim 22.

Secondary considerations

Applicant provides the following secondary considerations in order to persuade the Examiner that the claimed invention is non-obvious in view of the prior art.

The claimed invention has met a long felt but unresolved need to reduce development time and engineering cost for customizing embedded processor firmware for different applications and I/O devices. This is evidenced by the articles on the products covered by amended claim 22,

including the Amulet OS Chips and display modules including the Amulet OS Chips (collectively "Amulet products"). One article states:

> Consumers accustomed to seeing their electronic implements display lots of information on a snazzy LCD screen often don't appreciate how much work goes into creating the interface that makes it possible to do so.
>
> Graphic display panels are often treated as a sort of afterthought once a product design is almost complete. Presenting graphics may not be all that difficult for an Intel-made (nasdaq: INTC - news - people) Pentium chip in a PC. But for many electronic devices using much simpler 8-bit and 16-bit microcontroller chips, adding graphics with only a few fonts and icons can triple the size of the software code required to run the device. If the code gets too complex, the simpler chip ceases to be up to the task, and a more costly one has to be put in its place.
>
> Amulet's chip solves the problem by putting a complete graphical user interface on a separate inexpensive chip that can be easily programmed, and easily connected to most existing microcontroller chips. It's called the Easy GUI chip, and Amulet sells it as a standalone chip or installed directly on an LCD screen. Since Amulet is a fabless semiconductor company that doesn't have its own factory of "fab," Singapore's Chartered Semiconductor (nasdaq: CHRT - news - people) is building it for Amulet under contract.

"A Chip That Makes Electronics Easier To Use" by Arik Hesseldahl, Forbes.com, March 18, 2004 (emphasis added). Another articles states:

> A graphics panel is often an afterthought for embedded systems. In some cases, the main product doesn't have a graphics panel, but one can be purchased as an optional extra. I often see this in medical devices where, for example, numerical data representing a patient's condition can be seen on seven segment displays on the main device, while an optional add-on can take that data and display it in a graph. Another common practice is to not include a graphics panel on a current product, but call for one in the next generation of the design.
>
> Unfortunately, implementing a graphical user interface (GUI) is more than a matter of adding one more peripheral to the address/data bus and controlling it with the same microcontroller from the previous design. Graphics generally demand more processing speed and more code space. A device with ten buttons and a simple four-line LCD display might contain 32KB of code. A graphics display with a few fonts and bitmaps could easily add 100KB to the code size. Suddenly, that 8-bit microcontroller with on-chip flash and RAM is no longer up to the job.
>
> ....
>
> Serial module

> One option when you want to manage your graphics from a serial port is the Amulet Technologies GUI. This product contains an LCD, a touchscreen, and a small board to drive the LCD. In addition to managing the data and clock signals required by the LCD, the board provides a graphical library and some simple graphical objects.

"GUI add-on options" by Niall Murphy, Embedded Systems Programming, April 2, 2003 (emphasis added).

The Amulet products have been praised by those in the industry. The Amulet products have received awards including the Best New Product of the Year from Design News in 2003, and the Automation Excellence Award from the Instrumentation and Automation News in November, 2003.

The Amulet products have considerable commercial success. The Amulet products have been sold to over 40 customers. The Amulet products are currently being incorporated into products such as appliances, medical devices, gas station point-of-sale devices, home HVAC devices, water desalinization systems, ultrasonic equipment, and semiconductor manufacturing equipment.

The success of the Amulet products stems from the fact that the GUI controller operates the GUI independently from the embedded controller as recited in amended claim 22. This is evidenced by the articles on the claimed invention. One article states:

> Amulet's browser chip serves as both LCD controller and user interface engine. The intended architecture has the main embedded application running on a separate processor, communicating with Easy GUI via an RS-232 serial interface. This arrangement significantly boosts efficiency by freeing the main microprocessor and memory of any graphics work. An additional benefit is that much of the user interface logic and all of the text and images can be revised or upgraded with no changes to the main embedded device firmware.
>
> ....
>
> Interfaces for the Easy GUI are laid out in the same manner as an HTML website, a strategy clever in several ways. The standard and universal nature of HTML allows GUI prototype screens to be viewed in any web browser. Such accessibility makes the demonstration and testing of new interfaces as simple as visiting a web page. And advanced HTML editors like Macromedia Dreamweaver and Adobe GoLive can now be used to perform LCD interface design.

> Using tools included on a CD ROM, I successfully built and deployed a basic touch screen interface. Text, buttons, and images built up quickly as I followed useful examples in the MS Windows-based development kit. I compiled my new interface into native "µHTML" required by the display chip. Voila! In a single button click, my screens were up and running on the LCD display. The PC's standard 9 pin DIN "COM" port communicated over the supplied cable without a hitch.
>
> The Easy GUI compiler is competent in handling GIF and JPEG monochrome images as well as most HTML presentation tags. To achieve "flipbook" style animation for screen components, animated GIF images are rendered in motion and include speed control. A suite of scaleable proprietary "widgets" provides interface elements lacking in HTML, such as bar graphs, line plots, and sliders.
>
> ....
>
> Once an interface is compiled, uploaded, and running on the GUI controller, communication with the main process occurs over the serial port. Special codes embedded in the HTML invoke "widgets" to send and receive messages. Using this technique, a timer or user input from the touch panel initiates events. The Easy GUI client acts as master and can send five different types of messages including "get," "set," and a "remote procedure invoke." Included is a PC-based simulator allowing serial communications to be tested without a slave embedded device.
>
> While lacking the ultimate flexibility of traditional code-driven interfaces, Amulet's innovative use of HTML makes up the difference with simplicity and rapid development. With a list price of $399, Easy GUI is a unique system that should reduce labor and save time, particularly on embedded projects with basic to moderate screen demands. ....

"Tool saves time in LCD-interface design" by Jude DeMeis, Design News, April 8, 2002 (emphasis added). Another article states:

> One option when you want to manage your graphics from a serial port is the Amulet Technologies GUI. This product contains an LCD, a touchscreen, and a small board to drive the LCD. In addition to managing the data and clock signals required by the LCD, the board provides a graphical library and some simple graphical objects.
>
> You don't program this board in C. Instead, you load an HTML file to be stored in flash over the serial port. The serial port can then provide a conduit for messages that change values on the display. Such a message might be a temperature value sent to update the height of a bar. Messages received from the Amulet indicate when an event, such as a user pressing a button, has occurred.

The downloaded HTML file is not pure HTML, but the variations are slight. They call it HTML. While the format is similar to HTML, the user's experience is not mediated by a browser. The HTML format is simply a processor-independent way to describe the layout of a screen or a number of screens and locate text, bitmaps, and graphical widgets on those screens.

If you've defined a number of screens, navigating among them doesn't require any communication external to the Amulet. A user event, such as a changed value on a slider, will trigger communication. This minimizes the frequency of interruptions to the normal work of the microcontroller. In the other direction, the processor may transmit new values to the Amulet to update values, text, or widgets on the display.

This design creates a useful division between the control processor and the graphics engine. Since the a [sic] HTML file is the medium for graphics control, the graphics board also doesn't require any code. You have to program the control processor to perform some serial communications, but you don't have to bother with line-drawing routines and the like. More conventional graphics solutions offer a software library that you integrate with your own program. Amulet provides the low-level graphics as a hardware solution.

If you change the layout or redesign some icons, you just disconnects [sic] the graphics module from the controlling processor and moves [sic] it to a serial port of the PC. A new HTML file downloads, and now you've got your new user interface is in place. The upshot? Purely visual changes don't require modifications to the microcontroller software.

Since the Amulet interacts with the rest of the system via serial protocol, integration with a specific processor, RTOS, or compiler is not an issue. For this reason, the Amulet is by far the fastest design solution I have seen for adding a GUI to an existing system. ….

"GUI add-on options" by Niall Murphy, Embedded Systems Programming, April 2, 2003 (emphasis added).

Claims 23 to 34

Amended claim 23 recites a first controller that (1) decides on its own how to graphically present a parameter of a device independently from a second controller that controls the device, and (2) receives non-graphical data of the parameter from the second controller. As discussed above, Dye discloses an IMC that receives graphical commands from a CPU on how to graphically present windows/objects, and Margulis does not cure the deficiency of Dye. Accordingly, claim 23 is patentable over Dye and Margulis.

Claims 24 to 26 and 31 to 34 depend from claim 23 and are patentable over the cited references for at least some of the same reasons as claim 23.

Applicant has canceled claims 27 to 30, thereby rendering their rejections moot.

Claims 34 to 44

Amended claim 35 recites similar limitations as claims 22 and 23. Thus, claim 35 is patentable over Dye and Margulis for at least some of the same reasons as claim 23.

Claims 36, 37, and 41 to 44 depend from claim 35 and are patentable over the cited references for at least some of the same reasons as claim 35.

Applicant has canceled claims 37 to 40, thereby rendering their rejections moot.

Summary

In summary, claims 22 to 48 were pending when last examined where claims 45 to 48 were withdrawn from consideration. Applicant has amended claims 22, 23, 31, 33 to 35, 41, 43, and 44, and canceled claims 27 to 30 and 37 to 40. Applicant respectfully requests the allowance of claims 22 to 26, 31 to 36, and 41 to 44. Should the Examiner have any questions, the Examiner is invited to call the undersigned at (408) 382-0480.

I hereby certify that this correspondence is being deposited with the United States Postal Service as First Class Mail in an envelope addressed to: Mail Stop Amendment, Commissioner for Patents, P. O. Box 1450, Alexandria, VA 22313-1450, on the date shown below.

Signature: _____ Date: 1/30/06

Respectfully submitted,

David C. Hsia
Attorney for Applicant(s)
Reg. No. 46,235